COMMUNICATIONS AND INNOVATIONS GAZETTE

Proposta de uma biblioteca para replicação transparente em Sistemas Distribuídos utilizando JGroups

Caroline Martins Alves[1], **Odorico Machado Mendizabal**[1],

[1] Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
88.040-900 – Florianópolis – SC

caroline.martins@grad.ufsc.br, odorico.mendizabal@ufsc.br

Abstract. *The distributed systems development entails a series of challenges ranging from the heterogeneity of servers that host applications, through scalability and security, to the handling of failures. The present work focuses on the last aspect and, more precisely, on replication to provide fault tolerance, keeping system replicas available on different servers despite the occurrence of a limited number of failures. However, not all distributed systems implement this feature originally and, consequently, the developer is in charge of developing a replication logic suitable to the target system. This process can be expensive, time-consuming and error-prone, as it implies that the programmer has specific knowledge related to distributed systems and fault tolerance. Aiming to improve and to facilitate the use of replication in applications, the work being developed proposes the implementation of a library that offers replication in a transparent way for the programmer. Thus, systems that wish to apply this fault-tolerance strategy can easily do so, by just using our replication library. For the implementation of the library, JGroups will be used, a library for group communication.*

Resumo. *O desenvolvimento de sistemas distribuídos implica em uma série de desafios, que vão desde a heterogeneidade de servidores que hospedam aplicações, passando por escalabilidade e segurança, indo até o tratamento de falhas. O presente trabalho tem como foco o último aspecto e, mais precisamente, a replicação para oferecer tolerância a falhas, mantendo réplicas de um sistema disponíveis em diferentes servidores a despeito da ocorrência de um número limitado de falhas. Entretanto, nem todos os sistemas distribuídos implementam essa característica originalmente e, consequentemente é necessário que o desenvolvedor desenvolva a lógica de replicação especificamente para o sistema alvo. Esse processo pode ser caro, demorado e suscetível a erros, pois implica que o programador possua conhecimentos específicos relacionados a sistemas distribuídos e a tolerância a falhas. Visando melhorar e facilitar o uso de replicação em aplicações, o trabalho que está sendo desenvolvido propõe a implementação de uma biblioteca que ofereça replicação de forma transparente para o programador. Com isso, sistemas que desejem aplicar essa estratégia de tolerância a falhas podem facilmente fazê-la, apenas utilizando a nossa biblioteca. Para a implementação da biblioteca, será utilizado o JGroups, uma biblioteca para comunicação em grupo.*

1. Introdução

Com a evolução tecnológica, muitos serviços que antes só podiam ser executados fisicamente passaram a existir também no mundo digital, deixando as pessoas cada vez mais conectadas e, por vezes, dependendo dos serviços disponíveis na rede para trabalhar, para comunicar-se com família e amigos, para efetuar transações bancárias, para comércio eletrônico, entre outros. Essa mudança gerou uma grande praticidade aos usuários destes serviços, uma vez que diversas tarefas podem ser realizadas no conforto de suas casas, usando apenas um dispositivo conectado à Internet para acessar diferentes plataformas. Por outro lado, é necessário também que os provedores desses serviços garantam a sua disponibilidade e sua funcionalidade.

Quando um desses sistemas fica indisponível, acaba por gerar diversos transtornos e prejuízos para o usuário final, bem como para a empresa provedora. Para exemplificar essa situação, pode-se destacar alguns acontecimentos, como o GitHub, que, em fevereiro de 2020, esteve fora do ar diversas vezes, devido a variações inesperadas no carregamento do banco de dados e a problemas na configuração do mesmo [de Simone 2020]. Em agosto de 2021, o Banco do Brasil deixou cerca de 54 milhões de clientes sem acesso aos diversos serviços do banco [Globo 2021]. No mesmo ano, em outubro, as redes WhatsApp, Facebook, Instagram e Messenger ficaram fora do ar por quase 6 horas, causando um prejuízo de quase 6 bilhões de dólares para a Meta, empresa detentora dessas redes sociais [CNN 2021].

Nessa perspectiva, os sistemas distribuídos com ênfase em disponibilidade e em escalabilidade vêm ganhando espaço, permitindo o acesso crescente no número de usuários a um mesmo software com rapidez e com estabilidade. De acordo com Tanenbaum e Steen (2007), um sistema distribuído consiste em diversos computadores independentes, apresentados para o usuário como um único sistema coerente. Desse modo, o sistema encontra-se distribuído em diferentes computadores, aproveitando o poder de processamento de cada um destes, mas sem que o usuário perceba isso.

Entretanto, para reduzir os riscos e os impactos no caso de falhas de serviços distribuídos, espera-se que eles cumpram requisitos de sistemas confiáveis, como disponibilidade, confiabilidade, segurança e capacidade de manutenção [Verissimo and Rodrigues 2001]. Para assegurar que essas especificações sejam cumpridas, há várias técnicas de tratamento a falhas que buscam deixar os sistemas distribuídos o mais próximo possível de um sistema confiável, a exemplo da: ocultação de latência, replicação, ocultação de falhas, recuperação de falhas, entre outras.

A replicação, por sua vez, consiste em manter cópias de arquivos, de serviços ou até mesmo de sistemas inteiros, em diferentes servidores, de forma transparante para os usuários. Isso significa que, no caso de alguma réplica falhar, usuários ainda terão acesso aos dados ou à aplicação, ao conectar a uma das réplicas corretas. Implementar replicação exige coordenação entre as réplicas, uma vez que qualquer alteração feita em uma das cópias deve ser reproduzida nas demais [Charron-Bost et al. 2010]. Além disso, é necessário configurar o sistema com um número adequado de réplicas, para garantir o funcionamento correto da técnica. Portanto, a replicação não é facilmente implementada em sistemas distribuídos, necessitando que o programador implemente essa característica no sistema explicitamente. Esse processo pode ser custoso, já que requer que o desenvolvedor possua conhecimentos mais avançados em tolerância a falhas.

Visando sanar essa lacuna, este trabalho propõe a criação de uma biblioteca para replicação de serviços, de modo a prover tolerância a falhas. Consequentemente, sistemas que desejem prover replicação poderiam adicionar essa biblioteca ao seu software, sem programar do zero uma estratégia de replicação.

A seguir, na Seção 2, serão apresentados os principais conceitos e ferramentas em que esse trabalho está embasado. Na Seção 3, será abordado como pretende-se desenvolver a biblioteca, citando as tecnologias que serão utilizadas. A Seção 4 expõe alguns trabalhos que possuem correlação com a nossa proposta. Por fim, a Seção 5 apresenta as considerações finais e os próximos passos para a continuidade da pesquisa.

2. Fundamentação Teórica

Sistemas distribuídos são caracterizados por várias máquinas cooperando entre si, sem que o usuário final saiba da existência dessas diferentes máquinas ou como esse sistema opera internamente. Para isso, essa cooperação entre os diferentes computadores é essencial e, nesse sentido, tem-se que: "Um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente." [Tanenbaum and Steen 2007].

Para exemplificar esse cenário, pode-se imaginar a seguinte situação: um sistema de transações possui 3 computadores o disponibilizando de maneira íntegra, segura e confiável, em um dado momento, um desses computadores falha, porém ainda existem 2 computadores operando (Figura 1). Logo, o sistema continua disponível e o usuário continua o acessando normalmente. Isso só é possível devido à replicação, propriedade essencial desses sistemas e que será o objeto central desse trabalho. Graças a essa técnica de tolerância a falhas, não é necessário saber qual réplica está conectando ou realizar qualquer ajuste (alterar endereço, IP, porta, etc.), para continuar realizando as operações. Serviços oferecidos por bibliotecas de Comunicação em Grupo podem ser utilizados para implementar replicação, facilitando seu desenvolvimento e sua aplicação.

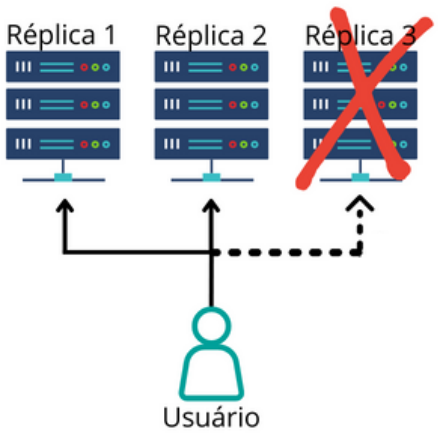

Figura 1. Abstração do funcionamento de um sistema distribuído

2.1. Comunicação em Grupo

Comunicação em grupo é definida por processos que cooperam a fim de sanar problemas de inconsistência durante a comunicação de sistemas distribuídos. Segundo

[Tanenbaum 1995], “Um grupo é um conjunto de processos cooperantes (que agem juntos) especificados pelo sistema ou por um usuário” e esses grupos podem receber e disseminar mensagens aos seus processos participantes, bem como enviar mensagens a demais grupos ou aplicações.

Assim, algumas das aplicações possíveis da comunicação em grupo são: realizar disseminação de mídias (a exemplo de um *chat*), além de tarefas mais complexas, como replicação ativa, garantia da entrega das mesmas ações em um contexto de banco de dados distribuído, entre outras.

A comunicação em grupo pode ter variações no seu tipo de comunicação, sendo *unicast*, quando as mensagens são trocadas no estilo ponto a ponto, no qual um participante de um grupo envia mensagens a um outro participante desse mesmo grupo, *multicast*, quando as mensagens são trocadas com um grupo de processos e *broadcast*, quando as mensagens são enviadas a todos os participantes do grupo. A Figura 2 apresenta o funcionamento dessas comunicações, ilustrando a comunicação um para um, um para todos e um para muitos, respectivamente.

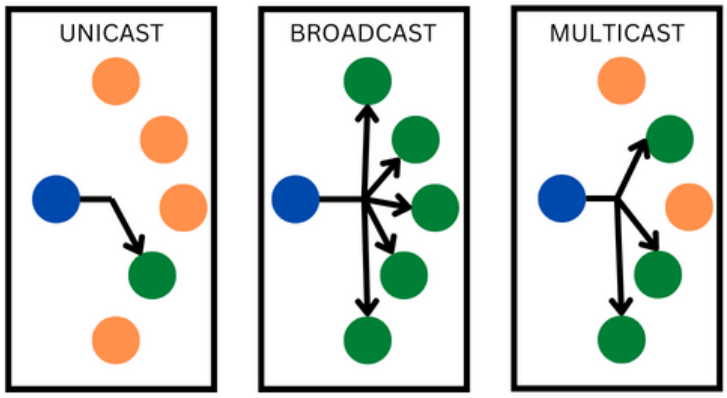

Figura 2. Tipos de comunicações no paradigma de Comunicação em Grupo (Adaptado de [TAHTEC 2020])

Em relação aos tipos de grupos, eles podem ser abertos ou fechados. Nos grupos abertos, há a interação com o ambiente que está fora do grupo, logo, aplicações externas podem enviar mensagens ao grupo e o grupo pode enviar mensagens a essas aplicações. Já nos grupos fechados, essa interação com o ambiente exterior não existe, havendo relacionamentos apenas dentro do grupo.

No quesito hierarquia, os grupos podem ser hierárquicos e não hierárquicos. No primeiro, os processos do grupo podem desempenhar diferentes papéis, como no caso de haver um nodo mestre, que controla determinadas ações e, demais nodos, que desempenham outras tarefas. No segundo, todos os participantes desempenham as mesmas funções e atividades no grupo.

Mais algumas características dos grupos são: em relação a sua dinâmica, permitindo que processos juntem-se a grupos existentes ou saiam de um grupo que fazem parte, em relação a sua confiabilidade, assegurando que a mensagem será entregue a todos os participantes ou, do contrário, será ignorada, respeitando o preceito de atomici-

dade - a mensagem chega a todos ou então a nenhum e em relação aos diferentes tipos de ordenações, que podem ser seguidos no momento da entrega dos processos, como FIFO (*First In First Out*), casual, total, entre outros.

Assim, este trabalho utilizará uma comunicação em *broadcast*, já que todos os participantes do grupo receberão as mensagens, fazendo as entregas destas com FIFO e utilizando um grupo aberto não hierárquico, pois o cliente (externo) terá interações com o grupo e todos participantes desempenharão mesma função. Este funcionamento ficará mais evidente na Seção 3, que detalha a estrutura da biblioteca.

2.2. JGroups

JGroups [Ban 2011] é uma biblioteca feita em Java que provê a troca de mensagem de maneira confiável. Com ela, é possível criar grupos com nós que podem trocar mensagens entre si. Algumas de suas principais funcionalidades são: criar e excluir grupos, entrar e sair de grupos, detecção de falhas em participantes de grupos, remoção de nós com falhas, envio e recebimento de mensagem em *unicast*, *multicast* e *broadcast*.

No contexto deste *framework*, existem dois conceitos que precisam ser entendidos, o de grupo e o de membro. Grupo é um *cluster*, processo hospedado em algum *host* e membro é um nodo que conecta-se a um grupo. Diversos nodos podem conectar-se a um *cluster* e o sistema mantém o controle dos membros que conectam ou que desconectam, notificando o *cluster* quando isso ocorre.

Já a arquitetura do JGroups, está organizada em: JChannel, *Building Blocks* e pilha de protocolos. O JChannel é usado pelos desenvolvedores para implementar aplicativos de comunicação em grupo confiáveis. Os *building blocks* ficam na camada acima do JChannel e proveem um nível maior de abstração. Já a pilha de protocolos, implementa propriedades específicas para um dado canal.

A Figura 3 representa essa estrutura, na qual o JGroups está organizado. Com isso, pode-se visualizar que um canal é conectado a uma pilha de protocolos: quando uma aplicação emite uma mensagem, o canal a transmite na pilha de protocolos, que, por sua vez, passa para o protocolo mais alto, processando a mensagem e enviando-a para o conseguinte. Logo, a mensagem é passada de protocolo a protocolo, até chegar no protocolo de transporte e ser publicada na rede.

Já o procedimento inverso ocorre para ouvir uma mensagem: o protocolo de transporte fica ouvindo a rede e, quando uma mensagem é recebida, ela é entregue para a pilha de protocolos, até chegar no canal. O canal aciona um *callback* na aplicação para entregar a mensagem. A pilha de protocolos é iniciada quando uma aplicação conecta no canal e, quando desconectar, a pilha será destruída, liberando espaço para outros recursos.

O JChannel é responsável por manusear e por controlar os grupos, permitindo desde sua criação até outras interações, que são viabilizadas pela API do JChannel. Para criar um grupo, é necessário definir um nome e todos os JChannels de mesmo nome formam um grupo, no qual seus participantes podem enviar e receber mensagens, sendo que os grupos não precisam ser criados previamente, ou seja, se um nodo liga-se a um grupo não existente, ele é automaticamente criado. Para sair do grupo, um participante pode fazer isso desconectando-se do canal e, devido sua característica de reuso, é possível que um membro que desconectou possa conectar, posteriormente, no canal. Entretanto,

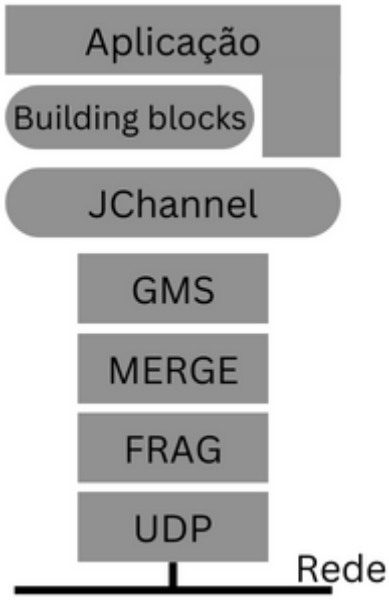

Figura 3. Arquitetura do JGroups (Adaptado de [JGroups 2002])

cada membro pode conectar apenas a um canal, não sendo possível fazer parte de mais de um canal ao mesmo tempo.

Os canais sabem quem são os membros dos grupos e, com isso, é possível gerar uma lista de endereços dos membros do canal, que é chamada de *view*. Também é viável que um processo direcione uma mensagem a um endereço da *view* ou que em *broadcast* envie uma mensagem a todos os membros, incluindo ele mesmo.

Ao invés dos canais, podem ser utilizados os *building blocks*, que ficam em uma camada acima deles e que são abstrações de nível mais alto, que concedem uma API mais sofisticada que o *JChannel*. Os blocos de construção podem criar e usar canais internos ou podem usar um canal existente para sua criação. Além disso, os *building blocks* podem oferecer acesso ao canal e, com isso, se o bloco não fornecer alguma funcionalidade, o canal pode ser acessado diretamente.

Outra propriedade muito importante da ferramenta é a sua pilha configurável de protocolos que permite a manipulação destes para atender diferentes necessidades, como particularidades de rede. Alguns dos protocolos disponíveis são: UDP (*User Datagram Protocol*), TCP (*Transmission Control Protocol*), controle de fluxo, detecção de falhas, criptografia, FIFO, além de ser possível escrever seu próprio protocolo.

Vale ainda ressaltar sua facilidade de uso e incorporação dentro de um código, sendo necessário apenas adicionar o .jar da biblioteca dentro do projeto para utilizá-lo. Ademais, a ferramenta conta com uma documentação detalhada [JGroups 2002], que permite consultar os diversos métodos da biblioteca e entender melhor seu uso e sua devida aplicação.

O JGroups é um *framework* robusto e muito bem consolidado, prova disso são as diversas empresas e soluções que o utilizam, a exemplo do JBoss, que usa o JGroups para implementação de funcionalidade de agrupamento no *JBoss J2EE Application Server*. Outro ponto é que ele incorpora de maneira excelente as premissas levantadas pela

Comunicação em Grupo, indo desde a difusão das mensagens, até níveis mais altos, como tratamento de falhas dentro do próprio grupo, e será um excelente acessório para o desenvolvimento deste projeto.

3. Desenvolvimento da Biblioteca de Replicação

A solução escolhida para implementar replicação de forma transparente foi inserir um *middleware* entre o cliente e a aplicação alvo, para que as devidas tratativas sejam feitas com o JGroups [Ban 2011] e, então, os dados sejam replicados. Logo, pretende-se desenvolver essa biblioteca, que funcionará como um interceptador, capturando pacotes enviados pelo cliente e replicando-os na aplicação alvo.

3.1. Estrutura da Ferramenta

Para que a implementação ocorra, foi necessário definir, primeiramente, sua estrutura e quais tecnologias seriam utilizadas. Será usado a ferramenta de comunicação em grupo JGroups [Ban 2011] para difusão de mensagens e para garantir todas as tratativas de confiabilidade que um sistema distribuído exige. Existem outras bibliotecas disponíveis no mercado que possuem esse mesmo propósito, a exemplo do Atomix [Foundation 2013], porém optou-se por utilizar o JGroups por ser um produto bem consolidado no mercado, sendo utilizado por servidores Tomcat JSP e JBoss J2EE [Abdellatif et al. 2004], contando com uma documentação bem detalhada, o que facilita seu uso. Optou-se pela linguagem de programação Java, uma vez que a ferramenta JGroups [Ban 2011] é escrita nela, então o uso dessa linguagem acaba por facilitar o desenvolvimento. Outro recurso escolhido foi o Java Socket [Oracle 2014], viabilizando a comunicação da biblioteca com o cliente e, posteriormente, com a aplicação alvo, que, por ser um pacote já bem difundido, é bastante prático e fácil de se utilizar.

Além disso, é essencial o uso de um padrão de projeto para a criação deste trabalho, pois esse artifício garante uma maior facilidade para manutenção e para extensão do código na adição de novas *features*. Padrões de projeto são soluções típicas para problemas comuns em projetos de softwares e, para implementação do interceptador, o padrão seguido será o embaixador [Microsoft 2022], que fornece uma maneira transparente de modificar ou de ampliar o processamento tradicional.

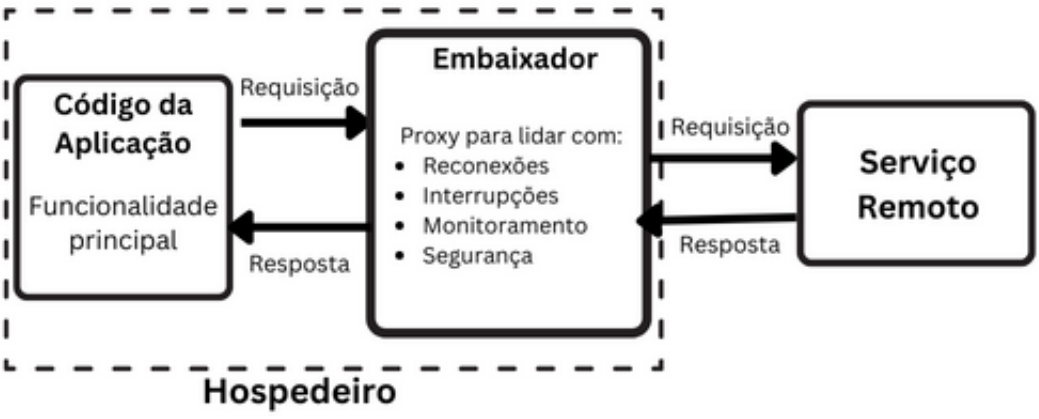

Figura 4. Exemplo do padrão de projeto embaixador (Adaptado de [Microsoft 2022])

A Figura 4 ilustra o padrão de projeto embaixador, que consiste em colocar uma estrutura (Embaixador), que será uma espécie de *proxy*, no mesmo hospedeiro que está localizado o código da aplicação, permitindo ao embaixador controlar roteamento, resiliência e recursos de segurança, dentre outras funcionalidades.

Já para a arquitetura da biblioteca que será desenvolvida, o conceito de *middleware* será adotado e, assim, um cliente enviará requisições a esta ferramenta de maneira transparente. Dentro da ferramenta será feito o uso de Comunicação em Grupo (por meio da biblioteca JGroups) e com isso será possível criar um grupo, com alguns participantes, que receberão a informação enviada pelo cliente e a entregarão nos destinos adequados, aplicando assim a replicação deste dado.

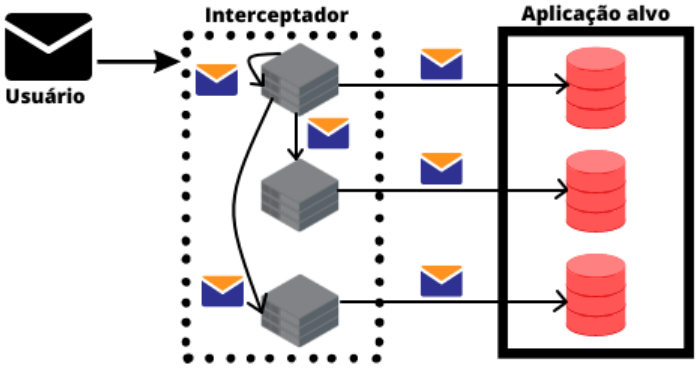

Figura 5. Funcionamento da estrutura da biblioteca

A Figura 5 ilustra o funcionamento esperado do *middleware*, em que o usuário emite uma mensagem com destino a uma aplicação, a biblioteca intercepta esse envio, capturando a mensagem e disseminando-a entre todos os processos participantes do grupo e, por fim, cada participante envia a mensagem a diferentes cópias da aplicação alvo, concretizando assim a replicação. Vale ressaltar que o interceptador será composto por n réplicas da biblioteca, que serão inicializadas e que estarão aguardando a emissão de uma requisição pelo cliente. Assume-se que até f réplicas podem falhar. Assim, o uso do padrão de projeto embaixador no desenvolvimento da biblioteca não cria um ponto único de falha, uma vez que haverá até $n - f$ réplicas ativas, sendo cada réplica responsável por uma instância replicada da aplicação alvo.

Esta arquitetura utilizada, mesmo provendo transparência de acesso entre usuário-interceptador, ainda possui algumas limitações. O cliente envia mensagens para o interceptador, sem precisar conhecê-lo, porém o interceptador precisa necessariamente conhecer os endereços da aplicação alvo para entregar essas mensagens. Além disso, até f instâncias do interceptador podem falhar e, consequentemente, estarão indisponíveis para os clientes, exigindo que o cliente descubra o endereço de outra réplica correta. Essas limitações, apesar de tornarem a solução menos transparente, não inviabilizam o seu desenvolvimento, uma vez que pode-se, por exemplo, contar com um arquivo auxiliar que mantém o controle dos endereços das réplicas e repassa ao cliente quais IPs de réplicas ativas para que as requisições sejam feitas. Alternativamente, um serviço de resolução de nomes pode ser utilizado em conjunto com a técnica.

3.2. Protótipos

Nos passos iniciais, foram desenvolvidas algumas aplicações para familiarizar-se com a ferramenta JGroups [Ban 2011], tornando possível apropriar-se de seus conceitos e métodos fundamentais. Algumas aplicações de teste foram criadas, uma delas utilizando Java e Spring Boot para a criação de *endpoints*, que serão necessários para que ocorra a comunicação do cliente com a biblioteca. Entretanto, como destacado na subseção 3.1, optou-se pelo uso de Sockets [Oracle 2014], já que a necessidade inicial é apenas fornecer um canal de comunicação do cliente com a biblioteca de replicação. Então, o uso da ferramenta Spring Boot tornou-se desnecessário, uma vez que Sockets fornecem essa mesma funcionalidade de maneira mais pontual. Essa escolha foi feita pensando em redução de custos computacionais, além de redução da complexidade da aplicação, já que o Spring Boot é uma ferramenta mais robusta e utilizada para diferentes propósitos que vão além da criação de *endpoints* e já o Sockets é mais enxuto nesse sentido.

Após o desenvolvimento de aplicações teste e familiarização com as ferramentas, foi possível dar início ao desenvolvimento da biblioteca. O que se tem, até agora, é uma aplicação que está ativa com um grupo de três participantes (este valor foi escolhido arbitrariamente e pode ser alterado posteriormente) aguardando receber uma solicitação de um cliente via Socket. Em um segundo momento, será desenvolvido o envio dessa solicitação do cliente a aplicação alvo por cada um dos três participantes do grupo.

A aplicação alvo escolhida para ser estudo de caso dessa solução é o banco de dados RocksDB [Facebook 2020], um banco em memória que não implementa replicação. Inicialmente, usaremos este banco para os primeiros testes e validações, mas a ideia do trabalho é chegar em um nível de generalização, em que a solução funcione independentemente de qual seja a aplicação alvo.

3.3. Teste e avaliação

Após o desenvolvimento da biblioteca estar completo, serão necessárias algumas verificações para confirmar se a ferramenta, além de resolver o problema inicial, o faz de maneira eficaz, sem gerar custos de desempenho demasiados que invalidem a execução da aplicação, por exemplo. É esperado que o uso da solução gere algum gasto computacional a mais, como o aumento da latência e vazão, uma vez que, ao invés do cliente entregar a mensagem diretamente ao destino, agora ela passa primeiramente pelo interceptador.

Para os testes, será utilizada a plataforma Emulab [White et al. 2002]. O Emulab é um ambiente de *testbed* que fornece o suporte necessário para alocar nodos e para configurar características de rede em Sistemas Distribuídos em ambiente controlado. Dentre os testes, algumas comparações devem ser feitas, a fim de validar a qualidade da solução, como, verificar o desempenho obtido quando uma aplicação utiliza a biblioteca e o desempenho quando a aplicação implementa replicação sem o suporte da biblioteca. Pretende-se instrumentar os serviços e os clientes para coleta de vazão de requisições processadas, bem como latência na resposta de requisições. Considerando execuções sucessivas, com acréscimo na carga de trabalho, deve-se produzir um gráfico que ilustre os valores de vazão e de latência, permitindo observar os pontos de saturação (*i.e.*, quando a vazão estabiliza e a latência passa a aumentar significativamente). A comparação de desempenho seguirá uma abordagem similar à proposta em [Pereira et al. 2018]. Não será estipulado um critério de aceitação, uma vez que o objetivo dos testes será avaliar o custo ocasionado

pela incorporação da biblioteca para replicação em uma aplicação.

4. Trabalhos relacionados

Nesta seção serão apresentados alguns trabalhos que possuem correlação com este projeto e propõem fornecer replicação de forma transparente, apontando para formas de implementar essa técnica de tolerância a falhas.

Em [Pereira et al. 2019], é apresentada uma estratégia para replicar trechos de código de um sistema, utilizando a abordagem de Replicação de Máquina de Estados (RME), por meio do protocolo de consenso Paxos [Lamport 2001]. Assim, para que o programador determine que certo método deve ser replicado, é necessário que utilize a anotação *@SmrMethod* (com seus devidos parâmetros), logo acima da assinatura do método. A ferramenta identificará, com base nas anotações de código, quais métodos devem ser replicados. Desse modo, o serviço de replicação oferecido pela biblioteca aumenta a disponibilidade ao custo de uma perda de desempenho. Além disso, foi observada uma perda de vazão ao utilizá-la, como já era esperado. Houve também um aumento da latência ocasionado pelo uso do protocolo Paxos [Lamport 2001] e não da biblioteca em si. Essa ferramenta possui um intuito muito parecido com o proposto neste trabalho, uma vez que os dois tratam de replicação em sistemas distribuídos de maneira transparente. Entretanto, a biblioteca de replicação de serviços exige que o programador anote os métodos que devem ser replicados, necessitando de uma intervenção do desenvolvedor e uma alteração da aplicação alvo para utilizá-la. Diferentemente, a nossa proposta visa integrar um serviço de replicação na forma de um interceptador, capturando o tráfego de requisições entre cliente e aplicação alvo. Espera-se que essa abordagem evite modificações na aplicação a ser replicada.

A proposta feita por [Ugliara et al. 2017] apresenta como usar a estrutura de meta programação da linguagem Cyan, em prol de desenvolver softwares replicados e tolerantes a falha por meio de anotações simples, sem que o programador precise saber detalhes profundos de replicação. Isso pode ser feito por meio de meta objetos, que podem ser anexados a métodos modificadores. O artigo mostra como fazer essa implementação, gerando e validando o código de integração que usa a estrutura de Treplica. Os autores também apresentam como a ferramenta faz verificações em relação a determinismo, na qual o desenvolvedor é alertado caso exista alguma operação inconsistente ou método modificador. Como trabalhos futuros, os autores pretendem criar um mecanismo capaz de substituir essas operações não determinísticas por operações determinísticas equivalentes. Assim como o trabalho apresentado em [Pereira et al. 2019], os autores propõem uma maneira facilitada de implementar replicação em sistemas. Entretanto, o artigo ainda exige que uma linguagem de programação (Cyan) específica seja utilizada para isso, enquanto este projeto almeja que a replicação seja possível de ser utilizada em diferentes contextos e sem a exigência de que seja feita em uma linguagem de programação específica.

5. Considerações Finais

Ao fim deste trabalho, espera-se obter uma ferramenta que atuará como um interceptador, replicando mensagens que são enviadas de um cliente com destino a alguma aplicação, utilizando inicialmente o RocksDB como caso de teste desta solução e, posteriormente, podendo expandi-la para demais aplicações.

Para validar o sucesso dessa biblioteca, serão realizados testes em ambiente de simulação de Sistemas Distribuídos, o Emulab, avaliando os custos agregados a essa ferramenta e quais os impactos causados por ela. Com isso, espera-se produzir avaliações (em formato de gráfico), que permitam visualizar e concluir qual o custo agregado causado pela solução em termos de vazão e de latência.

Atualmente, o que já foi desenvolvido foram pesquisas em torno dos temas base deste projeto, como Comunicação em Grupo e a própria ferramenta JGroups, além da avaliação do estado da arte nessa área (vide Seção 4). Assim, para finalização deste trabalho, o protótipo já desenvolvido precisa ser aprimorado e ter o seu desempenho avaliado, de modo a permitir ajustes e a finalização de sua implementação na forma de uma biblioteca de replicação.

Agradecimentos

O presente trabalho foi realizado com o apoio do Fundo Nacional de Desenvolvimento da Educação - FNDE e do PET Computação - UFSC.

Referências

Abdellatif, T., Cecchet, E., and Lachaize, R. (2004). Evaluation of a group communication middleware for clustered J2EE application servers. In *OTM Confederated International Conferences On the Move to Meaningful Internet Systems*, pages 1571–1589. Springer.

Ban, B. (2011). Reliable Multicasting with the JGroups Toolkit. http://www.jgroups.org/manual/html_single/. [Online; acessado em 15/12/2021].

Charron-Bost, B., Pedone, F., and Schiper, A. (2010). *Replication: Theory and Practice*, volume 5959. Springer.

CNN (2021). Mark Zuckerberg perde quase US$ 6 bi em dia de falhas e denúncias ao Facebook. https://www.cnnbrasil.com.br/business/mark-zuckerberg-perde-quase-us-6-bi-em-dia-de-falhas-e-denuncias-ao-facebook/. [Online; acessado em 04/03/2022].

de Simone, S. (2020). GitHub Was down Multiple Times Last February: Here's Why. https://www.infoq.com/news/2020/03/github-february-incidents. [Online; acessado em 25/07/2022].

Facebook (2020). RocksDB. http://rocksdb.org/. [Online; acessado em 25/07/2022].

Foundation, O. N. (2013). Atomix Framework. https://atomix.io/. [Online; acessado em 14/11/2022].

Globo, O. (2021). Sistema do BB volta a funcionar, após ficar sete horas fora do ar. https://oglobo.globo.com/economia/sistema-do-bb-volta-funcionar-apos-ficar-sete-horas-fora-do-ar-1-25174388. [Online; acessado em 24/07/2022].

JGroups (2002). JGroups - A Toolkit for Reliable Messaging. http://www.jgroups.org/. [Online; acessado em 23/07/2022].

Lamport, L. (2001). Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58.

Microsoft (2022). Padrão embaixador. https://docs.microsoft.com/pt-br/azure/architecture/patterns/ambassador. [Online; acessado em 25/07/2022].

Oracle (2014). Socket (Java Platform SE 8). https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html. [Online; acessado em 25/07/2022].

Pereira, P., Müller, R. H., and Mendizabal, O. M. (2018). Avaliação de desempenho de bibliotecas java para o protocolo paxos. In *Anais da XVIII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul*, Porto Alegre, RS, Brasil. SBC.

Pereira, P. M., Dotti, F. L., Meinhardt, C., and Mendizabal, O. M. (2019). A library for services transparent replication. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, page 268–275, New York, NY, USA. Association for Computing Machinery.

TAHTEC (2020). Diferença entre Unicast, multicast e broadcast. https://tahtec.com.br/diferenca-entre-unicast-multicast-e-broadcast/. [Online; acessado em 25/07/2022].

Tanenbaum, A. S. (1995). *Distributed Operating Systems*. Prentice Hall, New Jersey.

Tanenbaum, A. S. and Steen, M. V. (2007). *Sistemas Distribuídos: princípios e paradigmas*. Pearson Prentice Hall, São Paulo, 2. ed. edition.

Ugliara, F. A., Vieira, G. M. D., and de Oliveira Guimarães, J. (2017). Transparent replication using metaprogramming in cyan. In *Proceedings of the 21st Brazilian Symposium on Programming Languages*, SBLP 2017, New York, NY, USA. Association for Computing Machinery.

Verissimo, P. and Rodrigues, L. (2001). Distributed systems for system architects.

White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. (2002). An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Operating Systems Review*, 36(SI):255–270.